Vehicle System Dynamics
2004, Vol. 41, No. 5, pp. 415–429

Taylor & Francis
Taylor & Francis Group

Overview of Coupling of Multibody and Control Engineering Tools

ONDŘEJ VACULÍN[1,2], WOLF R. KRÜGER[2]
AND MICHAEL VALÁŠEK[3]

SUMMARY

The role of Computer Aided Engineering in vehicle development has been significantly increased during the last decade. Specialised simulation tools became very complex, however, growing demands on complexity and particularly interdisciplinarity of vehicles and their simulation models have led to a number of approaches trying either to develop multidisciplinary simulation tools or to connect various specialised simulation tools by interfaces. This paper addresses some aspects of interconnection of the specialised simulation tools as one possibility for simulating complex mechatronic vehicle systems. It classifies the interfaces between specialised software packages in general, mentions some historical development of the interfacing and further discusses the examples of the implemented couplings between the Multibody System codes and Computer Aided Control Engineering tools. Finally, the performance of selected interfaces is compared on an example simulation of a controlled vehicle suspension.

1. INTRODUCTION

Current vehicles and their components can be modelled with many targets, from different points of view, in several levels of complexity and using various Computer Aided Design (CAE) tools ranging from tools based on universal spreadsheets to very complex specialised tools. The increasing demands on vehicles together with relatively cheap and powerful electronics and actuator technology enable to extend purely mechanical systems with new control features. Such multidisciplinary systems together with requirements for shorter development times, lower development costs and new quality of products being developed have raised new requests on CAE tools.

Two basic strategies for modelling model the complex multidisciplinary vehicle systems can be distinguished. On one hand, the multidisciplinary systems can be

[1]Address correspondence to: Ondřej Vaculín, Institute of Aeroelasticity, Vehicle System Dynamics, DLR Oberpfaffenhofen, P.O. Box 1116, D-82230 Wessling, Germany. E-mail: Ondrej.Vaculin@DLR.de
[2]Institute of Aeroelasticity, Vehicle System Dynamics, DLR Oberpfaffenhofen, Germany.
[3]Department of Mechanics, Faculty of Mechanical Engineering, Czech Technical University in Prague, Czech Republic.

0042-3114/04/4105-415$22.00

modelled in one multidisciplinary environment often based on block oriented modelling or bond graphs. It means that all model components are implemented in a single environment. However, these tools do not usually offer effective specialised algorithms and particularly as large support of library elements as traditional specialised packages. This approach seems to be more widely used in the future, since the multidisciplinary tools are still in fast development.

Interfacing or coupling of specialised simulation tools is another of the possibilities for simulating the multidisciplinary models. In this case the models are prepared in specialised tools, which offer large support of library elements as well as intuitive modelling for the given sub problem. The paper addresses the interfaces in general, and coupling of CACE and MBS tools in particular.

The main CAE methods under consideration in this paper are Multibody Systems (MBS) and Computer Aided Control Engineering (CACE). However, the other CAE tools such as FEA (Finite Element Analysis), CFD (Computational Fluid Dynamics) and last but not least CAD (Computer Aided Design) can also contribute to the final multidisciplinary vehicle model.

2. CLASSIFICATION OF INTERFACES

Simulation tools have been originally designed as stand-alone applications. It is not very common for two simulation tools to use the same model data format (the same is true for most software engineering).

Several types of interfaces have been developed and implemented during recent years. The classification, which summarizes the most important features of the interfaces, seems to be necessary. Many different criteria for classifying the interfaces have been already presented in literature, for example [1–5]. Some classification categories are presented in the following paragraphs.

2.1. Work Flow

2.1.1. Uni-Directional and Bi-Directional Interfaces

If interfaces are evaluated from the work flow point of view a distinction can be made between uni-directional and bi-directional interfaces. A uni-directional interface is needed if one program is used as a pre-processor for a second program. Typical examples are grid generators for finite element analyses or codes generating symbolic code of a system to be included into a model and solved in another tool.

Bi-directional interfaces handle the flow of information between two running simulations. Consequently, the simulation results are at least partially available in both tools. Interfaces based on co-simulation are a typical example of this group.

2.2. Mathematical Physical Aspects

2.2.1. Model Description Level

For the classification of interfaces it is helpful to distinguish between different levels of model descriptions. Firstly, simulation models are often described with *application specific parameters*. In this case, only the class of a model element, often represented by a number or keyword, and values for pre-defined variables are given in a database file. The database input files contain certain library elements, for example a spring in an MBS code, which is referenced by its type-identifier and parameters. Such a description has the advantage that parameter-based input files are relatively short and of low complexity. However, the parameters in such input files do not give a lot of information about the underlying physical element definition. Interfaces based on such native model descriptions, and especially in the case of commercial packages changing the input file is often the simplest way to access these programs in an automated way.

The second level of the model description is the so-called *descriptive level*. In this case, a description of physical properties of the system as well as the parameters is defined. It includes particularly models described with sets of differential equations or transfer functions. The solution in time domain can be obtained by the connection of a model in descriptive level with an additional numerical solver. State-space matrices are a special case, that is linear time independent systems.

The so-called *operational models* represent the third description level of models. This level of models can directly generate new information about the structure and parameters of the model, for example its time behaviour. Thus, operational models can be realised as differential equations with a solver in order to generate the data about model behaviour in the time domain. An operational model can be a 'black box', meaning that the actual model properties are hidden from the user and only responses on defined inputs are given. Therefore operational models are common for interfaces, especially for co-simulation purposes.

2.2.2. Numeric Integration

Another classification of interfaces is based on their numerical integration schemes. The numerical integration of the coupled system can be performed in one tool by a common numerical integrator; this method is often called *tight* or *close coupling*. In this case, the sub-models have to be connected into one complete model and all the states of such a complete model have to be accessible by a single numerical integrator. Furthermore, the integrator must be able to handle all types of model behaviour and equations used by the included models.

The numeric integration can also be distributed. In this case the coupled tools each use their own solvers and only inputs and outputs are exchanged, most often at pre-defined communication time points. This scheme is often called (noniterative) *weak*

or *loose coupling*. The states of one sub-model are hidden for the numerical integrators of the other sub-models. Since both systems are solved separately by a special optimised numeric integrator, the performance could be increased; however, the communication intervals have to be chosen carefully as a trade-off between computational performance and numerical stability. Furthermore, it can be shown that some systems, for example with algebraic loops between the subsystems, do not converge at all with the noniterative loose coupling scheme and the iterative approach is proposed [3].

It should be noted at this point that both tight and weak coupling can be achieved independently from the selected implementation method. However, in practical applications the term co-simulation is often used exclusively for loose coupling in combination with a multi-process or inter-process communication (IPC) solution. In the following paragraphs, the term co-simulation will be used as a synonym for loose coupling in general.

2.2.3. Model Size Adaptation

Often models of different complexity are coupled. Differences are either the model size, for example the number of degrees of freedom, or in the type of system description. Many physical problems can be described, for example, dimensionless, in one, two, or three dimensions. If models of different complexity are coupled, solutions have to be found to either reduce the complexity of a sub-model to that of the main model or to interpolate between the sub-models. An example for model reduction are the use of modal representation of flexible bodies or the mathematical model reduction techniques used in control design; an example for the need of interpolation is the simultaneous use of 1D, 2D and 3D models in a turbine simulation.

2.3. Software, Hardware and Implementation Issues

2.3.1. Programming Technique

From the programming implementation point of view the interface can be realized as a *single process* or a *multi-process solution.* This classification is independent of the selected numerical integration aspect. Single processes can be obtained on the source code level or on the object code level. In the first case, source code is transferred and all sub-models or programs are compiled and linked into a single executable. The interface based on source code is platform independent. On the other hand it is possible to interchange pre-compiled objects and link them into a common executable. This can only be done, however, if all code modules have been compiled for the same platform and operating system. In a multi-process solution all models are simulated in their own executables and small-size code implementing the communication routines is added.

2.3.2. Data Transfer

In a coupled simulation the data have to be transferred between the sub-models. Data transfer can be performed inside a code by defined parameter lists of subroutine calls or between codes by file transfer, inter-process communication, or a mixture of both. The choice between the methods depends on the amount of information exchanged, performance, and the simulation environment available.

File interfaces are often used if models are results of pre-processors, have to be portable across platforms, and if a large amount of data has to be transferred between simulations. They are exported from one program and imported by the partner program. Inter-process communication (IPC) can be chosen if the processes run in parallel and the amount of data is not too large. The IPC processes can be connected by a network.

Inter-process communication in itself is a large field, and the selection of soft- and hardware is based on the requirements. Communication can be achieved by using directly basic functionalities of operating systems as shared memory or sockets, or by using more comprehensive commercial or public-domain packages which supply communication libraries as PVM, MPI, or CORBA, [6–8].

If large amounts of data have to be exchanged, for example in a coupled simulation of CFD and FEA programs, often file interfaces and IPC are used in parallel. Communication routines are used to schedule the process, but the bulk of the data describing a model is exchanged by files.

2.3.3. Platform Dependence

The coupling of simulations can be realized either on a single CPU as *single platform, single node*, or several computers of the same type (e.g., clusters) or on different nodes of the same computer *single platform, multi node*, or on different computers of different types and/or operating systems *multi platform*. All these variations require different solutions for simulation interfaces.

For complex work flows comprising several programs on distributed networks a number of specialized coupling libraries (e.g., CORBA) and work flow managers, addressing the questions mentioned above, have been developed.

3. HISTORICAL OVERVIEW

The previous paragraphs have opened a wide range of possibilities for the interfacing of simulation tools. Historically, the demand for interfaces came from the multibody code users and developers. It was driven by the necessity to design and simulate new mechatronic products where the controller complexity increased as well as by need to include elastic bodies into MBS. The export of linearised models from the MBS packages to the CACE tools is a relatively simple task, but such an interface did not satisfy all demands on the coupling anymore.

The MBS codes were historically better prepared for the model export than CACE, and since the activity came from the groups developing multibody packages, it was also simpler to adapt the MBS code in order to be able to communicate with CACE packages.

Originally, in the early nineties the tight coupling, that is numerical integration of the whole system by one integrator, was preferred because of foreseen stability problems of weak-coupling (co-simulation) and also available computer power. Still, the numerical integrators in CACE tools did not show the performance of the specialised integrators in the MBS tools. The tight coupling was realised both on the descriptive model basis and on the operational model basis, in both cases as single process solutions.

Later the co-simulation interfaces came, which opened questions of inter-process communication. The main motivation was to enable simulation of complex MBS models, which could not be handled by tight coupling interfaces and/or to increase simulation speed. It was a small step from single process solution to the IPC.

All mentioned versions were transferring MBS model to CACE tool or were based on the co-simulation. The so-called "inverse" interfaces, which transfer control structures to the MBS, came in the last years of the previous century. The goal was to simplify controller design in MBS package and furthermore to use more powerful and robust numerical apparatus available in MBS packages.

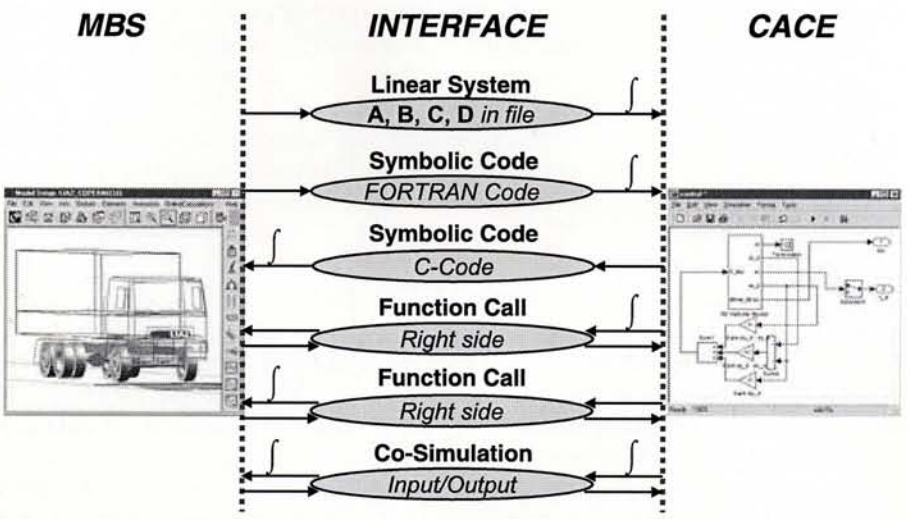

Fig. 1. Typical example of interfaces between MBS and CACE tools.

The current developments are performed under new objectives such as software support and maintenance. The interfaces, which are based IPC strategies or code-export with precompiled libraries, are preferred.

Figure 1 presents a typical example of a family of interfaces between MBS and CACE tools. The arrows indicate the direction of data flow, that is right sides or inputs/outputs. The sign '$\int$' defines on which side the numerical integration is performed.

4. EXAMPLES OF INTERFACES

The MBS package SIMPACK, [9], has one of the most extensive sets of interfaces, and for this reason is used as an example to illustrate the possible options, although other MBS packages such have some comparable features [10, 11]. Among the interfaces to the other CAE tools, SIMPACK offers also interfaces to CACE tools such as MATRIXx/SystemBuild and MATLAB/Simulink.

During last ten years several types of SIMAX and SIMAT interfaces has been implemented to couple SIMPACK with CACE tools:

- Linear System Interface,
- Symbolic Code Interface,
- Function Call Interface,
- Single Process Co-Simulation Interface,
- IPC Co-Simulation Interface,
- Inverse Symbolic Code Interface,
- MBS Syntax Interface.

4.1. Linear System Interface

One possibility to describe general linear systems is by linear system matrices **A**, **B**, **C**, **D** for state space representation of the system dynamics in the form of

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u},$$
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}.$$

SIMPACK can directly generate time-invariant matrices **A**, **B**, **C**, **D** for a given linearisation state to a text file readable by CACE tools. The Linear System Interface is classified as descriptive model file interface with tight coupling implemented as a single process.

The Linear System Interface is suitable for linear system analysis and control design. Systems represented by the matrices are easily interchangeable not only between program packages, but also between different computer platforms.

4.2. Symbolic Code Interface

Nonlinear multibody systems can be described by sets of nonlinear differential equations. A symbolic code generated by SIMPACK is an optimised FORTRAN code representing the nonlinear differential equations describing a specific multibody model, that is descriptive models.

The symbolic code has to be linked and compiled together with an interface allowing the generated code to be called in the CACE convention as a user defined dynamic subsystem, for example Simulink S-function. Finally, the SIMPACK model is represented by one block with corresponding inputs and outputs in a SystemBuild or Simulink block diagram. Similarly to the Linear System Interface, the Symbolic Code Interface is a file interface working with descriptive models, is implemented as single process tight coupling, but accounts for the full nonlinear dynamic model behaviour.

The Symbolic Code Interface is independent of a computer platform, but if the multibody system is modified, the FORTRAN code must be generated, compiled and linked again.

4.3. Function Call Interface

Similarly to the Symbolic Code Interface the Function Call Interface generates a set of nonlinear differential equations of the multibody system and the whole system is numerically integrated by the CACE tool (tight coupling). In this case the generic SIMPACK database file with application specific parameters, in which the multibody system is described, is used.

Because of operational model transfer, no additional files need to be generated by SIMPACK and no compilation is necessary if the model is changed. It enables a SIMPACK MBS model to be quickly tuned in interaction with Simulink simulation. The Function Call Interface even enables to create the data files for SIMPACK post-processing including 3D animation during the SystemBuild or Simulink simulation. It makes the interface bi-directional.

The Function Call Interface is also created as a user defined dynamic subsystem (single process) in the CACE tool, but in this case, the interface includes all SIMPACK libraries, in which SIMPACK elements, equation generation and file handling are programmed. Figure 2 presents the implementation of the Function Call Interface with MATLAB/Simulink.

SIMPACK and the CACE tool must run on the same platform, or at least on a SIMPACK supported system.

The current developments are focused on the function call interface, which uses high performance of SIMPACK numerics. The tight multi-process connection between MATLAB and SIMPACK is performed by MATLAB ENGINE.

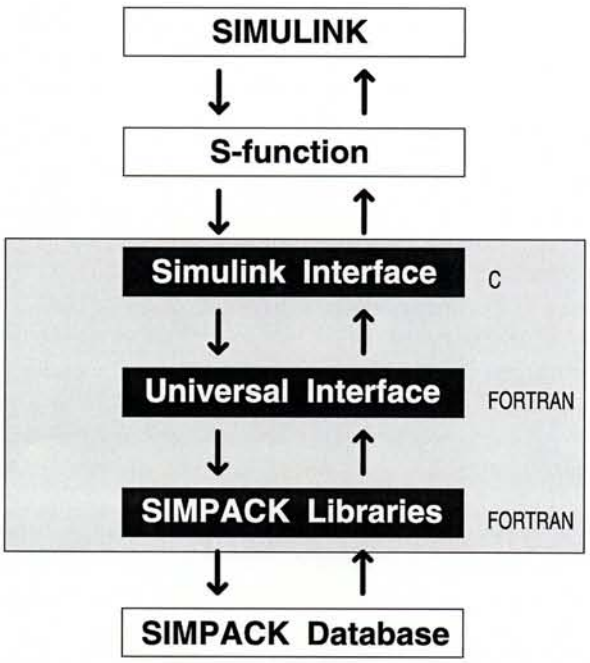

Fig. 2. Structure of the SIMAT Function Call Interface.

4.4. Co-simulation Interfaces

Both co-simulation interfaces are based on operational models with weak coupling. It means that both codes, SIMPACK as well as the CACE tool, provide the numerical integration and communicate with each other in time steps, which are in both cases fixed. Between these time points SIMPACK provides the continuous integration using inputs obtained from CACE tool and returns new outputs at the new time step, and vice versa, CACE tool numerically integrates its part using the inputs obtained from SIMPACK in the last time point.

Since all MBS model components are solved inside SIMPACK, no restrictions are applied to the modelling. The interface is capable of using models in the differential algebraic equation formulation where the state vector can include rigid body states, elastic body states, force element states, holonomic constraints and other algebraic equations to determine additional auxiliary conditions (e.g., for the on-line determination of accelerations and of friction forces). As in the Function Call Interface, all simulation results are available for post-processing in both SIMPACK and CACE tool. The restrictions are that through the use of sequential

("step-by-step") co-simulation system stability can often only be reached by relatively small communication intervals and, for some cases, stability is not even theoretically guaranteed. However, the small communication intervals decrease the simulation speed.

4.4.1. Single Process Solution

The single process solution has been developed for MATLAB/Simulink as an extension of the Function Call Interface because MATLAB/Simulink was not able to handle models described with differential algebraic equation.

The Single Process Co-Simulation Interface is also created as a CACE user defined dynamic subsystem (S-Function), which includes SIMPACK numeric libraries with numeric integrators. The Simulink numeric integration is the master, which considers the SIMPACK model block as discrete and communicates with it in periodic fixed time steps.

4.4.2. Two Process Solution

A combined simulation of SIMPACK and the CACE tool can be performed using co-simulation via inter-process communication (IPC). For this possibility SIMPACK and the CACE tool do not need to run on the same computer. Each package forms its own executable which communicates by the means of sockets, i.e. a network link providing a two-way communication channel between processes, either user-programmed or based on commercial or public-domain IPC libraries. Data exchange is performed in discrete fixed time steps between the two processes.

4.5. Inverse Symbolic Code Interface

The Inverse Symbolic Code Interface includes the CACE model into the SIMPACK model. The CACE subsystem is created in the CACE environment. The C-source code and interface files are generated with the aid of AutoCode Export (MATRIXx) or Real Time Workshop (MATLAB), compiled and linked with SIMPACK libraries. A new SIMPACK user element, which contains the original CACE model, is created (single process) and can be used in SIMPACK models. Furthermore, the SIMPACK user has access to the model parameters defined in the original CACE subsystem. The interface generates the code, which is independent of the CACE installation. Thus the CACE subsystem – SIMPACK user element can be created on other platforms than the one SIMPACK is running on and licensed for.

4.6. MBS Syntax Interface

MBS Syntax Interface offers the exchange of parameters in the application specific parameter level. Sometimes the result of a CACE calculation is only couple of

parameters for which the Inverse Symbolic Code Interface export would be too cumbersome. In this case it is possible to save the results of the control design or parameter optimisation in the syntax of single SIMPACK elements. An element thus defined is then placed in the data base from which the simulation model is assembled by special script files.

5. COMPARISON OF INTERFACES

The family of interfaces contains several members. Each interface has its specific function and fulfils certain requirements. If the mechatronic design is required, the interface must enable modification of both MBS and CACE part of the model. The application of symbolic code interfaces seems in such a case rather inconvenient because of the necessary recompilation of the model after its modification. Or, if a mechanical model described with differential algebraic equation is to be coupled with the CACE, one should be careful about numerical integrators available in CACE tools.

In order to compare the efficiency of Function Call (FC) Interface, Single Process Co-Simulation (CoSi) Interface and Co-Simulation Interface with Inter-Process Communication (IPC) a simple simulation example is presented. The FC Interface uses the numerical integration in MATLAB/Simulink (tight coupling) and the CoSi Interface numerically integrates the MBS and CACE parts separately (weak coupling). The IPC Co-Simulation Interface is also based on the weak coupling, but is implemented as multi(two)-process solution with IPC.

5.1. Simulation Model

Since the numerical integration of systems described with differential algebraic equation is not straightforward in Simulink, the SIMPACK model should be described with ordinary differential equations; the mechanical model should not contain any algebraic constraints, for example caused by closed kinematic loops.

The mechanical model of order 38 (Fig. 3) represents a platform truck, which is equipped with semi-active dampers on the driven axle. The semi-active dampers are controlled by an extended groundhook controller, [12], implemented in MATLAB/ Simulink. The model was designed to optimise semi-active control algorithms for road-friendly trucks [13]. The original purpose of the model was to investigate the vertical dynamics, because the vertical dynamics plays a significant role in road-tyre forces generation, but in this paper it is used to provide a comparison between the performances of different kinds of interfaces.

Two different road surfaces are used to excite the model: (i) a deterministic ramp 0.08 m high and 5.8 m long with ascent and descent lengths 2.5 m, [14], and (ii) a

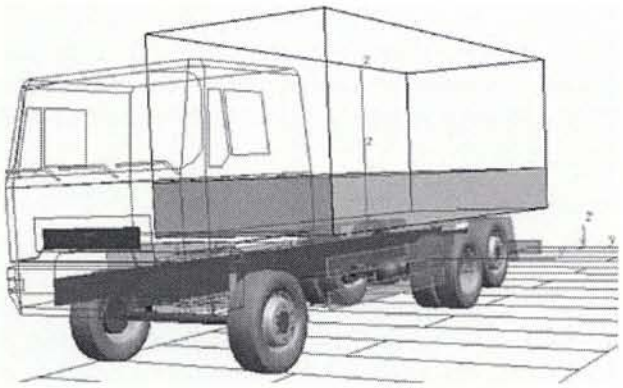

Fig. 3. Vehicle model.

stochastic road. In both cases the simulation time is set to 5 s and the initial velocity is 20 m/s.

5.2. Simulation Results

The comparison of the performance has been performed on a PC based two-processor computer (Intel XEON 1.7 GHz, 768 MB RAM) with MS Windows XP operation system. This hardware configuration should be an advantage for the multi-process IPC Interface, because SIMPACK can run on one processor and MATLAB/Simulink on the other one. To get the results also for more traditional one-processor computer for the multi-process IPC Interface, the second processor was suppressed. The influence of the two-processor architecture compared to one-processor architecture is marginal for the single process interfaces (FCI and CoSi).

The model has been simulated in seven simulation environments:

1. Function Call Interface (FCI)
2. Single process Co-Simulation (CoSi) with communication step $T = 0.01$ s
3. Multiprocess IPC Co-Simulation on both processors (MIPC) with $T = 0.01$ s
4. Multiprocess IPC Co-Simulation on single processor (SIPC) with $T = 0.01$ s
5. Single process Co-Simulation (CoSi) with $T = 0.002$ s
6. Multiprocess IPC Co-Simulation on both processors (MIPC) $T = 0.002$ s
7. Multiprocess IPC Co-Simulation on single processor (SIPC) $T = 0.002$ s

In order to get average values, the model has been simulated five times in each simulation environment. The normalised average simulation times are presented in Figure 4 as well as in Table 1. The FCI simulation times are used as a basis for the

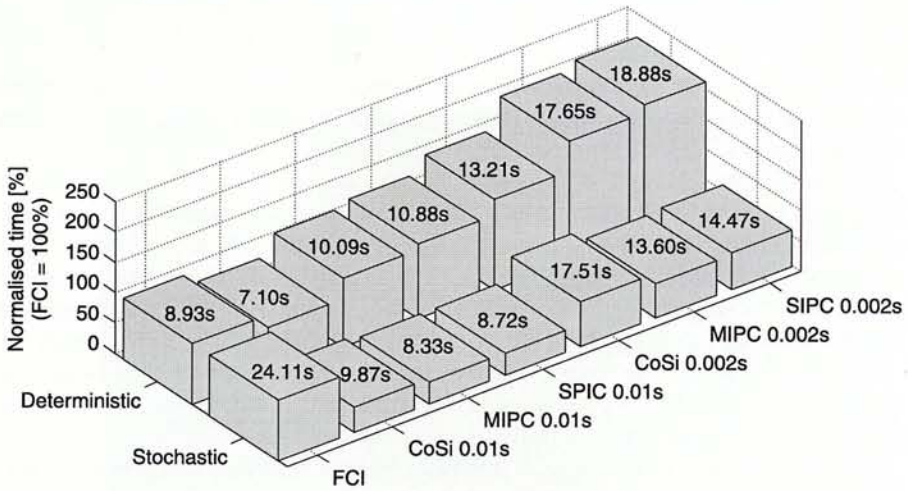

Fig. 4. Simulation times.

Table 1. Comparison of normalised simulation times (FCI = 100%).

Interface	FCI	CoSi	MIPC	SIPC	CoSi	MIPC	SIPC
T [s]	–	0.01	0.01	0.01	0.002	0.002	0.002
Deterministic	100	80	113	122	148	198	211
Stochastic	100	41	35	36	73	56	60

normalisation both for deterministic and stochastic road. Moreover Figure 4 presents the absolute simulation times.

The results indicate that the model with the deterministic road is faster, if numerically integrated in one environment. The integrator can control the step size in a relatively free range. The co-simulation interfaces are limited by the fixed communication steps, which seems to be too short for some phases of this simulation scenario. However, in the case of the stochastic road, the co-simulation seems to be an advantage, because the steps remain relatively small and the mechanical part is numerically integrated with an optimised integrator. The two-processor architecture accelerates the computation by about 5 to 7%. It is caused by the different complexity of the SIMPACK and MATLAB/Simulink models, i.e. the computer spends significantly more time with integrating SIMPACK model than MATLAB/Simulink one. The differences between single (CoSi) and two-process (MIPC and SIPC) solution are caused by the communication overhead of the IPC solution, which is independent of the model complexity but depends on the sampling rate.

6. CONCLUSIONS

The classification of interfaces has been introduced. Various interfaces, which connect MBS and CACE tools have been presented. The family of interfaces contains several members, each of which has its specific function.

The simulation example showed that the selection of the interface depends on many factors. The selected system, truck with controllable damping, is not very sensitive to instable behaviour, because no energy is added to the whole system. Thus the simulation time is of prime interest. The example has shown that too small communication step can significantly decrease the performance of the simulation and that the communication overhead plays a considerable role for simple scenarios.

Since the hardware implementation of the controller is usually performed by digital controllers with defined constant sample rate, the weak coupling with the same sample rate as future digital controller is of advantage.

The development of MBS/CACE interfaces has not been concluded with the IPC interfaces. Further development will include numerical aspects of co-simulation, for example co-simulation interfaces with variable time steps for data exchange, code generation for real-time applications and model optimisation for special cases such as hardware in the loop applications.

REFERENCES

1. Kortüm, W.: Trends in der Rechnersimulation im Hinblick auf die Systemdynamik Fahrzeug – Fahrweg. In: *Proceedings of the VDI, Simulation und Simulatoren für den Schienenverkehr*, München, 1995, pp. 169–203.
2. Veitl, A., Gordon, T., van de Sand, A., Howell, M., Valášek, M., Vaculín, O. and Steinbauer, P.: Methodologies for Coupling Models and Codes in Mechatronic System Analysis and Design. In: *Proceedings of the 16th IAVSD Symposium on Dynamics of Vehicles on Roads and Tracks*, Pretoria, 1999, pp. 231–243.
3. Kübler, R. and Schiehlen, W.: Modular Simulation in Multibody System Dynamics. *Multibody Syst. Dyn.* 4 (2000), pp. 107–127.
4. Vaculín, O., Krüger, W.R. and Spieck, M.: Coupling of Multibody and Control Simulation Tools for the Design of Mechatronic Systems. In: *Proceedings of the ASME 2001 Design Engineering Technical Conferences, DETC2001/VIB-21323*, Pittsburgh, 2001.
5. Krüger, W.R., Vaculín, O. and Kortüm, W.: Multi-Disciplinary Simulation of Vehicle System Dynamics. In: *RTO Meeting Proceedings 89: Reduction of Military Vehicle Acquisition Time and Cost through Advanced Modelling and Virtual Simulation*, Paris, 2002, pp. 16(1)–16(16).
6. Beguelin, A., Dongarra, J., Geist, A., Manchek, R., Moore, K. and Sunderam, V.: Tools for Heterogeneous Network Computing. In: *SIAM Conference on Parallel Computing*, Norfolk, Virginia, March 1993.
7. Snir, M., Otto, S., Huss-Lederman, S., Walker, D. and Dongarra, J.: *MPI: The Complete Reference*. The MIT Press, Cambridge, MA, 1996.
8. Rodríguez, J.I., Jiménez, J.M., Funes, F.J. and García de Jalón, J.: Dynamic Simulation of Multi-Body Systems on Internet Using CORBA, Java and XML. *Multibody Syst. Dyn.* 10 (2003), pp. 177–199.

9. SIMPACK: http://www.simpack.com/, 2003.
10. MSC.Software Corporation: http://www.mscsoftware.com/, 2003.
11. LMS International: http://www.lms.be/, 2003.
12. Valášek, M., Novák, M., Šika, Z. and Vaculín, O.: Extended Ground-Hook – New Concept of Semi-Active Control of Truck's Suspension. *Vehicle Syst. Dyn.* 27 (1997), pp. 289–303.
13. Vaculín, O.: *Multi-Objective Suppression of Vehicle Suspension Vibration.* PhD thesis, Czech Technical University in Prague, 2000.
14. Valášek, M., Stejskal, V., Šika, Z., Vaculín, O. and Kovanda, J.: Dynamic Model of Truck for Suspension Control. In: *Proceedings of the 15th IAVSD Symposium on Dynamics of Vehicles on Roads and Tracks*, Budapest, 1997, pp. 496–505.